# Initial Approaches for Discovery of Undocumented Functionality in FPGAs

***Matthew French, Andrew Schmidt, and Aravind Dasu***
{mfrench, aschmidt, adasu}@isi.edu
Information Sciences Institute
University of Southern California
Arlington, Virginia, USA, 2203

**Abstract:** *Due to a variety of commercial pressures such as IP protection, support cost, and time to market, modern COTS devices contain many functions that are not exposed to the end user. This creates a risk in government systems that undocumented hardware functions could become a security vulnerability. Traditional approaches require imaging a device and reverse engineering its functionality, a time consuming process which requires access to costly capital equipment. In this feasibility study, we present an approach tailored for FPGAs which leverages run-time active probing to greatly reduce the cost and time to discover undocumented functionality. We conduct an analysis on the Xilinx Virtex-5 DSP48E unit for which we identify the functionality for 1,136 undocumented modes.*

**Keywords:** FPGA; Undocumented functionality; Knowledge-based partitioning; On-chip testing; Partial reconfiguration; Sub-circuit extraction;

## Introduction

Modern integrated circuit devices have become enormously complex, both in sheer size and architecture, as heterogeneous System-on-Chip devices are commonplace. Commercial Field Programmable Gates Arrays (FPGAs) are at the forefront of these trends with over 10 billion transistors, and over 15 types of heterogeneous Hard IP blocks available to an end user, and several more which only the vendors are aware of.

These undisclosed features have become more prevalent in the sub 65nm fabrication era. As fabrication costs have escalated with each node and time to market pressures have increased, industry increasingly uses the current generation device to do trial runs of next-generation architecture features. IC vendors also seek to reduce overhead costs by re-using masks for similar products but enabling different features through software support or packaging. In addition, there are many built in self tests and other yield diagnostics that are not exposed to the end user.

Undocumented features are the product of industry operating in a highly cost competitive market, and are not inserted with malicious intent, however, this does not preclude that they could become a vulnerability which is exploited by others. A prominent example is the JTAG backdoor discovered in an Actel FPGA [1]. Therefore it is important that the full functional capabilities of a COTS device be known, in order to properly evaluate the potential security risks to DoD systems.

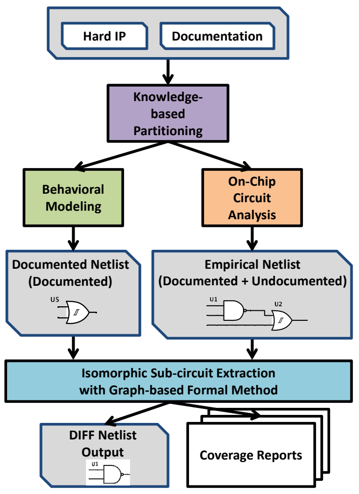
Hard IP
Documentation
Knowledge-based Partitioning
Behavioral Modeling
On-Chip Circuit Analysis
Documented Netlist (Documented)
U5
Empirical Netlist (Documented + Undocumented)
U1
U2
Isomorphic Sub-circuit Extraction with Graph-based Formal Method
DIFF Netlist Output
U1
Coverage Reports

**Figure 1** Functional Discovery Tool Suite

## Overview of Approach

We define functionality as a logic level operation that an end user of a device can access via programming, if they were given full authorization to do so. Documentation is defined as all the functionality described in user's guides for the product. As an example, if a multiplier circuit was found to have an accumulation function which was not described in the user's guide and was accessible via conventional, but undescribed, modalities, this would be an undocumented function. Producing novel behavior from a device through unconventional manipulation (eg lowering or raising the voltage outside of recommended ranges) is out of scope. With these definitions, our goal is to derive a user-centric representation of the functionality, not the specific way that the circuit was implemented. To achieve these goals our approach leverages several key insights:

- The FPGA programming abstraction level is described at a hardware level, unlike CPUs and GPUs, resulting in less variance between user documentation and underlying functionality and implementation.
- FPGAs have rich circuit level documentation in user's guides and patents that can bootstrap knowledge about the underlying circuitry.
- FPGA programmability creates controllability / observability properties, not typically available in ASICs, which can be utilized to configure and probe undocumented states using custom tools.

Our approach is depicted in Figure 1. The first step is Knowledge-based Partitioning where we accumulate knowledge bases and segment the design for further analysis. Based on open documentation, we develop a functional model of the IP in the Behavioral Modeling stage. The On-chip Circuit Analysis stage then probes the device during run-time and updates the original behavioral model to include any discovered functionality. The Isomorphic Sub-circuit Extraction stage then compares the resulting two netlists and removes equivalent circuits, leaving a circuit level representation of the undocumented functions. The next section provides an overview of the Xilinx Virtex5 DSP48E IP block under investigation to provide the reader a concrete example in order to better understand the technical approach.

## DSP48E Hard IP Overview

The DSP48Es in the Xilinx Virtex-5 devices [2], Figure 2, is the hard IP under investigation for this study. The DSP48E block has a long heritage with increasing capabilities across FPGA families, and is of a moderate sized complexity for a reasonably sized study. The Virtex-5 DSP block documented functionality includes multiplication, multiply and accumulate (MACC), three-input add, barrel shift, wide-bus multiplexing, magnitude comparator, bit-wise logic functions, pattern detection, and wide counters.

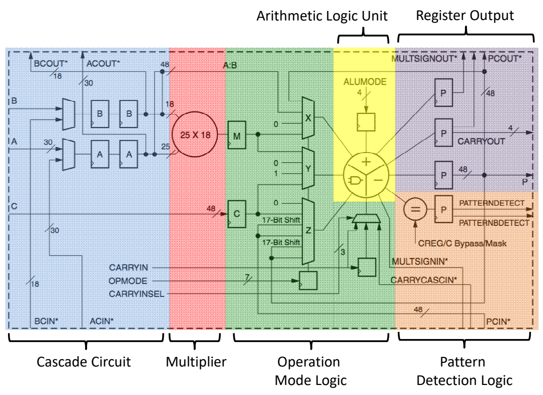

**Figure 2** DSP48 with Knowledge-based Partitions

The DSP48E user primitive has 335 input/output signals which include both control and data signals. A majority of these signals include the A, B, and C input operands and P output resultant. A simple analysis of the user's guide [2] reveals there are a plethora of undescribed states in the Cascade circuit and the Operating Mode logic. The Cascade circuit, which is set during configuration, only documents behavior for the REG and CASCREG for sets (0,0), (1,1), (2,1) and (2,2). The five other settings, such as (0,1), are undescribed. In the Operating Mode logic, which is set during runtime, all settings for the Z OPMODE of "111" are declared illegal, as well as 512 Y OPMODEs and 268 X OPMODEs. In our analysis, 1518 of the 2048 possible modes for the DSP are not documented.

## Knowledge-based Partitioning

In the knowledge based partitioning step the IP block is manually subdivided into units that can be further reduced in complexity, shown in Figure 1. This is accomplished through the analysis of documentation, user guides, patents, or any vendor provided simulation models which suggest sub-block functionality of the IP. FPGA documentation is largely provided at the circuit level, so it is easily decomposable into viable sub-blocks, from which documented behavioral models are constructed. The approach is well suited for FPGAs which are developed with modularity in tile type (Slice, BlockRAM, DSP etc...). For the DSP48E, the raw functionality is broken down into the following atomic units: cascade circuity, 25-bit x 18-bit multiplier, operation mode logic, arithmetic logic unit, register output, and pattern detection logic, as highlighted.

## Behavioral Modeling

Once the atomic sub-blocks have been identified, behavioral models are created. Presently, this is a manually process which involves constructing VHDL and Verilog simulation models, independent of any vendor models. Due to the hierarchical decomposition available and overall size of this IP block, manual coding is trivial. More complex circuits will may need to leverage generic open source cores or direct truth table translations. Since the behavioral models rely on documented information describing the functionality, the model is intended only to capture valid modes specified by the vendor. Test vectors are used to validate the model using commodity tools to perform automated test pattern generation. The model is synthesized using Synopsys Design Compiler to generate a netlist to be used by the isomorphic sub-circuit extraction stage.

## On-chip circuit Analysis

On-chip circuit analysis is used to selectively configure, probe, and analyze the empirical behavior of the IP. A suite of tools has been developed to identify undocumented bitstream configurations for the IP block and to provide run-time testing on an actual device. The bitstream configuration settings for the IP are isolated to determine if any additional parameters are possible beyond what is suggested in the IP's user guide. For example if the user guide described 7 states, this requires 3-bits ($2^3$=8 states), leaving one state unaccounted for. The tool flow identifies the missing state and generates a bitstream for on-chip testing to compare against the behavior model.

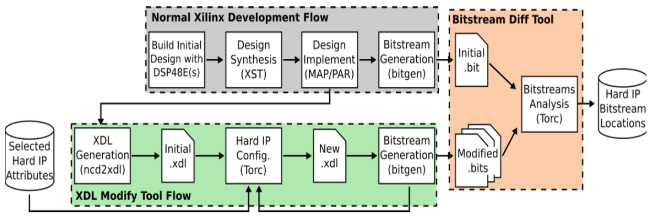
Normal Xilinx Development Flow
Build Initial Design with DSP48E(s)
Design Synthesis (XST)
Design Implement (MAP/PAR)
Bitstream Generation (bitgen)
Bitstream Diff Tool
Initial .bit
Bitstreams Analysis (Torc)
Hard IP Bitstream Locations
Selected Hard IP Attributes
XDL Generation (ncd2xdl)
Initial .xdl
Hard IP Config. (Torc)
New .xdl
Bitstream Generation (bitgen)
Modified .bits
XDL Modify Tool Flow

**Figure 3** On-Chip circuit Analysis Tool Flow

The on-chip circuit analysis tool flow, Figure 3, consists of the normal Xilinx Development Flow (Synthesis, Implementation, and BitGen) to create an initial bitstream which performs the original design behavior. The design can also be modified after the Implementation stage leveraging Torc [3] to change the Hard IP block's configuration attributes. Once the modifications are made BitGen is again performed to create new modified bitstreams. The Bitstream Diff tool is then used to compare the different bitstreams to identify which bits are not covered by the available configurations of the IP block. These locations are stored for further analysis during the on-chip run-time testing.

Knowledge based partitioning also provides undocumented modes from the IP block's datasheets, user guides, and patents. A majority of these modes are configurable at runtime, requiring a sophisticated on-chip testing infrastructure and testing methodology. The on-chip testing leverages active partial reconfiguration to selectively re-configure just the IP block under test to accelerate the overall testing. During each test, run-time data is collected to provide insight into the outputs of the experiment. For the DSP48 block the outputs of the product and accum register are stored by the MicroBlaze processor in memory. This run-time data is collected and analyzed through functionality scripts to determine whether the probed behavior matches the expected hypothesized behavior or is unexpected behavior. The behavioral model developed during the Knowledge-based partitioning is then updated to reflect the changes based on the on-chip testing to generate the empirical model. These two models are then used in the Isomorphic Sub-circuit Extraction stage.

## Isomorphic Sub-circuit Extraction

For this IP, most functionality is readily identifiable from the output of the On-chip Circuit Analysis stage. However, it can still be useful to have a circuit level description of the undocumented functionality, and larger more complex circuits may require automation to identify unusual functionality. The Isomorphic sub-circuit extraction stage compares the original behavioral model against the functional representation of the On-chip Circuit Analysis output, removes common circuits, and leaves a model of the undocumented functionality. This removes significant state space, making the resulting circuit easier to analyze. Towards this, a graph mining algorithm and tools have been developed to search for instances of known fundamental-circuit structures (such as multiplexers) in a larger netlist (such as the cascade circuit, or Operation model logic in a DSP module), in order to achieve state space reduction and formal verification.

The algorithm consumes two netlists: (a) The fundamental module/circuit to be mined, for example a 2:1 Mux in the form of a synthesized Verilog netlist, and (b) a large netlist that is expected to contain one or more instances of the fundamental module. For example, this can be a component of the DSP48E of the Virtex-5 FPGA, such as the cascade component, which contains three instances of a 2:1 Mux along with several registers and other control circuits. The synthesized netlist of the fundamental module (termed as the 'small' netlist/graph) is initially seeded, by selecting the net with the largest connectivity. Often this is the net with the largest fan-out. This seed, or initial sub-graph, is then grown into larger sub-graphs, by applying a set of instructions: Add cells, Add nets, and Connect nets. Through this process, the sequence of instructions and the resulting suite of sub-graphs are recorded/memorized for processing the larger netlist.

Next, the algorithm seeks to find identical copies of the seed net in the large netlist/graph. This initial search seeks wires with the same number of connections, regardless of the gates/std-cells that it connects to. Next, the sequence of growth instructions previously memorized, is applied in a formal verification loop. I.E. each time the tuple of instructions (add cells, add nets, and connect cells) is applied on the potentially-identical- seeds of the larger netlist, the sub-graph (as a Verilog netlist) is compared against the peer sub-graph from the small netlist.

The comparison is performed using Synopsys Formality. A caveat is that since Formality requires explicit binding of either input ports, or output ports prior to a formal verification process, the algorithm involves an implicit port binding process. A second caveat is that Formality, primarily intended for minor circuit changes (engineering change orders), is being used for entirely different purposes. This poses a challenge of mimicking routine Formality user practices, such as absorbing inverters at outputs or inputs to mitigate logic inversions. The algorithm uses an implicit process (a Python script) to automatically explore or discard the process of inverter absorption.

If the formal verification process passes the candidate sub-graph, then it is inspected for isomorphism. This implies that a perfect and complete match has been found in the large netlist. If this check yields an incomplete match, the algorithm continues to iterate through the sub-graph growth process, until either a match is found or the growth stalls

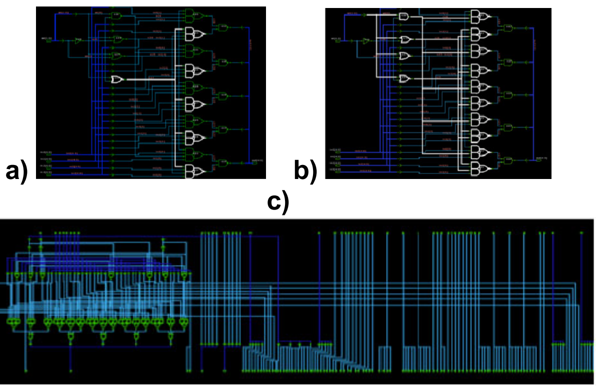

**Figure 4.** a) 4:1 seed net b) Sub-graph after two growth iterations c) Resulting mined ALU-input control circuit

due to a complete failure to grow any further. At this point, the mining process stops and the matched subgraphs (partial or isomorphic) are deleted from the large netlist. This process reduces the state space of the large netlist, thus allowing for either a small state space based manual/alternate inspection for functional mismatch, or subsequent mining of other fundamental modules. The algorithm terminates by generating a report of the candidates mined, and coverage obtained.

As an illustrative example, we consider the ALU-input control (Green in Figure 2) and its synthesized version, using Synopsys Design Compiler. The behavioral model used to generate the large netlist, leveraged the discovery of the undocumented features via the on-chip testing methodology and knowledge-enhancement from relevant Xilinx patents. Specifically, the discrete distributions of the control signals to the three multiplexers (X, Y, Z) and the redundant case of 110 and 111 for the select lines of the Z-Mux were considered. The small netlist used to reduce the complexity of the larger circuit was then a 4:1 Mux. Figure 4 depicts the seed net, the growth sequence, and the resulting mined circuit.

## Results

Using the developed on-chip testing infrastructure a total of 1136 out of the 1518 undocumented modes have been tested on the DSP48E block of a Virtex5 FPGA. This covers 74.8% of the undocumented states, as seen in Table 1. The remaining 25.2%, predominately on the Z Operand Multiplexer, have been intentionally ignored as the scope of this effort was only to evaluate 33.3% of the undocumented modes and was not due to any underlying technical approach issue. The techniques developed as part of this effort could be used to finish evaluating the unevaluated modes as part of future work.

This study uncovered several interesting results. To briefly summarize them, the cascade circuit was originally considered to only consist of five undocumented modes. However, through bitstream analysis two additional undocumented modes were discovered, resulting in seven evaluated undocumented modes for each of the A and B cascade circuits. In addition to the cascade register, the ALU and Operation mode circuit analysis identified the capability of extracting partial product outputs from the multiplier, intermediate shift register values, and even internal constants used by the circuit to perform Boolean logic operations. These are all likely self-test functions that the vendor did not see a reason to disclose to users and are not strictly malicious or illegal; however, they are clear examples of real functionality not being fully disclosed by the vendor. The full results are over 36 pages long and can be found at [4].

**Table 1** Undocumented Modes

| | Knowledge Based Partitioning Identified Undocumented Modes | On-Chip Circuit Analy Evaluated Undocumented Mo |
|---|---|---|
| **Cascade Register:** | | |
| A Input sub-circuit | 5 | |
| B Input sub-circuit | 5 | |
| **Op/ALU Modes:** | | |
| X Operand Multiplexer | 268 | [illegible] |
| Y Operand Multiplexer | 512 | [illegible] |
| Z Operand Multiplexer | 728 | [illegible] |
| Register Output | 0 | |
| Pattern Detection Logic | 0 | |
| Multiplier | 0 | |
| **Total:** | **1518** | **[illegible]** |

## Conclusion

In this work we have shown that there are promising approaches for detection and classification of undocumented functionality in FPGAs. This approach benefits from the level of documentation and observability and controllability inherent in FPGA devices. Further research and development is warranted to evaluate other FPGA IP. Software approaches such as these can be utilized to focus and prioritize the amount of silicon that needs to be inspected through time and capital intensive imagery techniques.

## Acknowledgements

This material is based upon work supported by the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001-11-C-0041. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

## References

[1] S. Skorobogatov, C. Woods, "Breakthrough silicon scanning discovers backdoor in military chip," Cryptographic Hardware and Embedded Systems Workshop, September 2012.

[2] Xilinx, Inc., "Virtex-5 FPGA XtremeDSP Design Considerations User Guide (UG193) v3.5," January 2012.

[3] N. Steiner, A. Wood, H. Shojaei, J. Couch, P. Athanas, M. French, "Torc: Tools for Open Reconfigurable Computing," 19th ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, February, 2011.

[4] M. French, A. Schmidt, and A. Dasu, "ITAG FPGA Undocumented Functional Discovery Final Report," September 2015.